SDN-RADAR: Network Troubleshooting Combining User Experience and SDN Capabilities

Gabriela Gheorghe, Tigran Avanesov, Maria-Rita Palattella, Thomas Engel
SnT, Université du Luxembourg,
Email: first-name.last-name@uni.lu

Ciprian Popoviciu
Nephos6 Inc.,
Email: chip@nephos6.com

***Abstract*—Software-defined deployments are growing into data center and enterprise network infrastructures. The typical promises of software-defined networks (SDN) are improved time for market, decreased risk and operational costs for services, flexibility and unified management. However, little is known and shared about how to actually manage an SDN network, especially in localising underperforming network paths (what we call "troubleshooting"). We describe a novel approach to ease large network troubleshooting by leveraging SDN features and incorporating distributed monitoring of network traffic. We suggest SDN-RADAR, a tool that can help network administrators understand which is the most likely faulty network link. To the best of our knowledge this is the first troubleshooting solution that combines user-side performance measurements with network data extracted from the SDN controller.**

I. Introduction

For massively distributed services, infrastructure failures can often damage user experience. When failures happen, service degradation costs money. For example, downtime in a U.S.-data center is worth more than $5000 per minute [1]. Unfortunately, locating failures in distributed infrastructures that span multiple domains and locations is not an easy task. First, the chaining of network protocols makes it difficult to understand the cause of a fault. A typical infrastructure (e.g., Wikipedia) spans several locations, DNS servers, load balancers, caching servers, replicated databases, and several webservers. Should there be delays in web content delivery or should the content be stale, it would be difficult for a network administrator to find out where in the protocol chain (HTTP, DNS, etc) the problem originates. Second, debugging tools are typically linked to certain protocols. Many IT administrators of traditional networks are still using *ping*, *traceroute* and *tcpdump*, as noticed in [2], [3], but this is not scalable for large networks with multiple protocols (e.g., IPv4 and IPv6) that can even be encapsulated one into another. Third, network troubleshooting is hard because traffic can be forwarded over different/multiple paths in the network; forwarding devices and end-user devices are heterogeneous, or can be virtualised.

With the spread of SDN deployments, it is natural to ask if managing and troubleshooting (or'debugging') future infrastructures (fully, or partially SDN-enabled) will become more challenging than it already is. At this point it becomes highly relevant to tackle questions such as: what can be reused from existing network management tools in an SDN network? How can SDN features help in network troubleshooting? To answer such questions, in this paper we are proposing a solution that can help administrators of large SDN-enabled networks monitor and debug network faults that affect service delivery on the user side. Our intention is to help building an early warning system that can raise flags if certain paths taken by application traffic are underperforming. Information about what actually happens with traffic within the infrastructure (for example, paths taken, changes in the headers, etc.) is missing in current troubleshooting technologies, and our view is that such information becomes very valuable when combined with network configuration and topology data to quantify the size and scope of the network issue that has occurred. With SDN capabilities, such a warning system can be optimised because of the north-bound features that the controller is offering: in particular, switch topology information, hosts database and a unique interface to query switches. This approach is also valuable in that it facilitates the transition to SDN, as it can help operationalize and optimize hybrid environments.

The contributions of our work are twofold. First, this paper proposes a general and flexible architecture to monitor and locate network performance issues, by leveraging SDN features to identify the most probable underperforming links in the network. Our solution integrates a distributed monitoring infrastructure that can be extended or replaced by more sophisticated monitoring tools. Second, we have developed a prototype of the proposed design, using state of the art SDN technologies in the open-source community (i.e., OpenDaylight, OpenVSwitch and Mininet) A heuristic method to indicate probable underperforming links is suggested, and a fomalisation of this method is also given.

II. Related work

Traditional network troubleshooting. As previously observed [4], [3], network troubleshooting is generally still in a very early stage. "Rudimentary" probing tools like *ping*, *traceroute*, *tcpdump*, or *nmap* statistics are still being used for detecting network issues, but they are insufficient for large complex infrastructures with high amounts of traffic.

Network troubleshooting (troubles are packet drops, delays, or reordering) has been approached with a range of packet marking techniques. X-Trace [5] traces multiple applications across different administrative domains, and at different network layers. Net-Replay [6] focuses on recovering from issues

This publication is based in part on work performed in the framework of the CoSDN project, INTER/POLLUX/12/4434480, funded by the Fonds National de la Recherche Luxembourg, and partly by the GEN6 EU-project 297239.

978-1-4799-7899-1/15/$31.00

by diagnosis and selective packet replay. Such approaches require the modification of clients, servers, and network devices to add protocol headers where relevant trace information is kept. Netcheck [7] diagnoses network problems based on traces derived from syscall invocations from multiple hosts; Netcheck only relies on a general model of a network, but is not aware of network topology and device state or configuration. Automated test packet generation [2] is a technique to automatically generate a minimal set of packets to test network liveness (at the level of links) and reachability (in terms of rules installed in network devices), for a given topology and configuration. This approach is somehow akin to ours in that monitoring traffic is injected into the monitored network, but additionally SDN-RADAR focuses on actual user-perceived performance.

SDN troubleshooting. Network Debugger (NDB) is a generalised tracing tool that resembles *tcpdump* but has knowledge of paths [8]. NDB cannot diagnose performance bugs, but bugs related to the correctness of forwarding (e.g., logical errors, switch implementation errors, packet format errors). Troubleshooting packet drops for different protocols, forwarding loops, and congestion points based on packet histories is done by NetSight [9]. NetSight reconstructs packet history by combining network topology with "postcards", events that are sent back to a debug point every time a packet traverses a switch. Another black-box network debugging approach, OFRewind [10], suggests a record-and-replay technique by which to infer, in retrospect, those OpenFlow events that generated a bug. OFRewind is more an network experimentation technique, while SDN-RADAR's approach is live monitoring of user experience. A comprehensive survey of SDN networks was recently made by Kreutz et al. [4]. The authors acknowledged it is still hard to tell what happens to packets in the network between source and destination, and how they are modified by network devices.

We can conclude that few if any related works concentrate on monitoring end-user experience as a starting point for troubleshooting. Most of the current troubleshooting approaches incur high costs in terms of network instrumentation, scalability and maintenance. More effort should be invested in understanding where to start looking for faults based on their effects, as well as in correlating perceived service degradation, with possible causes.

III. Sonar and distributed monitoring

Network management and troubleshooting tools are not aware of how far the service infrastructure is from satisfying service delivery needs, at any time. We give two real examples to illustrate this point. First, the networking team at Louisiana State University (LSU) acknowledged that traditional network management tools cannot inform about service reachability across various services. The value of this information became apparent when their autonomous system was blacklisted for IPv6 communication based on Google's assessment of poor user experience over IPv6 for some users within the LSU network. Traditional tools could not help identify and address the issue. Second, in the case of a large infrastructure such as the elections presentation service [11], a network outage or DDoS attack would require that a human administrator correlate a large set of log data, from several locations, and rely on intuition and experience to understand what the issue can be, what segment of users it affects, and its scope in the system. Such examples show that existing tools fall short in the face of complex network troubleshooting needs. Current technologies are not focused on users (many IT professionals and decision makers are complaining about the lack of user experience visibility), have vendor lock-in, usually are expensive and require heavy instrumentation of the network [4].

v6Sonar or simply Sonar [12] is a platform that implements the concept of distributed monitoring and troubleshooting. Sonar agents are scripts with a small footprint (i.e., consuming very little local resources) that are platform independent. They are configurable and collect the key data needed in calculating synthetic user experience for specific services. The agents are deployed on small embedded platforms or on existing network infrastructure equipment and are managed by a controller designed to interface with other controllers and orchestrators (e.g. OpenStack http://openstack.org). Agents are dynamically managed by the controller and can be assigned complex workflows mimicking a typical user workflow (for example, a set of website interactions). In this way, they can convey valuable information on how users experience an online service.

IV. Troubleshooting with SDN-RADAR

With the present work, we aim to help network administrators quickly determine the location and type of a network fault. Our approach to troubleshooting is unique in that it starts from a user perspective and progressively works its way towards the source of the fault. Normally the way users experience a web-based distributed service should be homogeneous from all locations. However, users experience the service conditioned by the quality of the underlying network links, which varies across domains and communication protocols. It is possible to put a threshold over the acceptable end-user perceived degradation, and start troubleshooting once such a threshold is surpassed. The choosing of such a threshold value can stem from an existing service level agreement (SLA). SLAs are usually specifically tailored to the service context and define limits of service delivery quality. For example, the average speed to answer a request is a common metric; some SLAs in VoIP contexts can refer to notions such as the "expectation factor"[1] — the amount of service degradation that users accept in service quality, in exchange for service access. Since the agents offered by the Sonar tool were already available, our solution is novel in that it combines Sonar's continuous monitoring capability, with the central and local network knowledge delivered by the SDN infrastructure, in order to assist in the fault localisation process.

SDN-RADAR is designed as a runtime application for network troubleshooting that "gets the pulse" of a target service from the point of view of user experience from various locations. When there is a drop in user experience from a particular location, the troubleshooting application seeks to locate the fault by retrieving the path actually taken by test packets from that location, and calculates which are the most likely segments on that path where the fault occurred. The architecture shown in Figure 1 consists of several elements:

[1]Cisco IOS IP SLAs Configuration Guide handbook, http://www.cisco.com/c/en/us/td/docs/ios/12_4/ip_sla/configuration/guide/hsla_c/hsvoipj.pdf

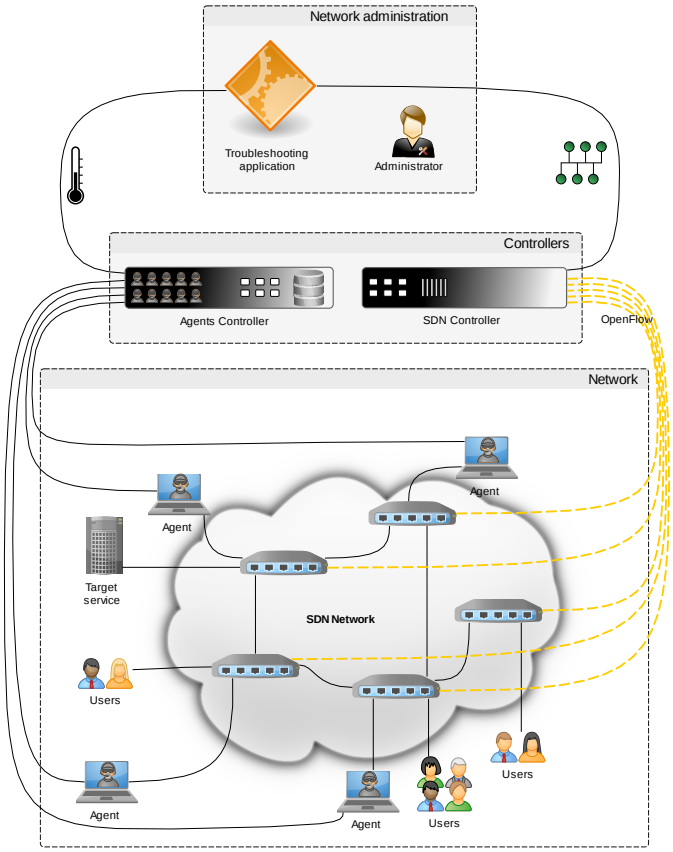

Fig. 1. SDN-RADAR architecture.

Target service. The target service is the set of infrastructure elements responsible for delivering the end service to users. In most cases the service delivering infrastructure consists of: multiple servers, load-balancers, proxies, firewalls, etc.

Agent. The agent essentially represents a user of the target service, and measures user experience over the target service. It is connected to switches in the network and periodically measures a set of user experience parameters.

Agents controller. The agents controller coordinates the work of agents and collects their reports. The controller can request to the agents to monitor a specific target, or a set of targets. It can also specify what kind of measurements the agent(s) should perform and their periodicity. The reported measurement values are stored in the controller's database.

SDN Controller and switches. Actual OpenFlow-based SDN network devices.

Troubleshooting application. This application that takes into account the measurements reported by the agents (via the agents controller) and any other features provided by SDN controller (like network topology) to identify problematic links in the network. The results can be displayed to the network administrators allowing them to take further measures for improving the network performance.

Agents give the administrator an idea about the degradation of end-user experience experienced from different locations. The agent information tells the administrator how big is the drop, and which are the locations to experience it. It would be useful to understand what is happening to the packets from those locations on their way to the target service and back, what path these packets take, and what modifications their headers underwent on the way. With the centralised management design of SDN networks, the administrator can easily retrieve network topology information and understand the paths taken by packets at certain locations.

V. SDN-RADAR Implementation

We implemented the proof of concept for the described architecture (Figure 1) [14]. Mininet [15], a simulated network environment was used to obtain an openflow network with hosts As a controller we used OpenDaylight [16] in the Hydrogen release. The communication with the troubleshooting app which is written in Python is performed using the ODL REST API. Monitoring agents run at different locations in the network and perform periodic measurements, such as round trip time (RTT) of a packet until it reaches the target host represented by a single host in Mininet. In the prototype the agents measure RTT to the target every 5 seconds from their network location. We used RabbitMQ (http://www.rabbitmq.com) and JSON messages for communication to the the troubleshooting application.

Knowing the topology is important to map the agents to the network diagram which allows one to better interpret the data that agent X produces. We can detect which path in the network topology is used to transmit packets from X to the monitored service by querying switches. Once we know the path, we associate the measurement with it and we store it for comparison. Both topology and unified interface to the switches are available in ODL.

Aiming to provide a tool for administrators to perform network troubleshooting, we have developed a graphical application interface. The troubleshooting application has a Python-based backend and web frontend that displays the network topology, location of the agents, their current measurements as well as network paths taken by the packets from a given source to the target host. The interface also allows the user to set different hosts as targets of agent monitoring and to set the threshold that will separate the reported measurements into acceptable or non-acceptable ones ((e.g., RTT < 250ms). Depending on the measurements, the agents that report them are marked with red, or green color (representing unacceptable or acceptable measurement values). For the convenience, the information about hosts and links extracted from the SDN controller is also available via the application interface. When the troubleshooting is performed, each link suspected of underperforming is automatically assigned a weight, indicating to the user where to start a more thorough analysis.

1) Faulty link detection algorithm: To detect a faulty link along an underperforming path, we have designed an algorithm that associates weights to network links, in such a way that a bigger weight means a higher probability of a failure point. The algorithm aims to discover the links shared by several paths: the more underperforming paths a link belongs to, the higher the probability that the detected service degradation occurs on this link. As such, the algorithm will associate a higher weight to this link. Also, if the link belongs to a problem-free path, we establish that link does not cause problems, so it will get the smallest weight (0).

To formalize the network context and to describe our algorithm, we use a multi-graph structure for the network. In what follows we introduce some basic notions of the model.

Nodes and connectors. Let V be the set of network nodes (hosts, switches). Each node $v \in V$ can have multiple ports enumerated with integers (i.e., enumerated switch ports). We will call a pair node-port a *connector* — it uniquely identifies a connection point in the network. For example, if we have a switch v with 16 ports, its connectors will be (v, i), where $i = 1, 2, \ldots, 16$. The set of all connectors of nodes in V is marked as C_V.

Link is a pair of two connectors. Specifically, $((v, 1), (u, 2))$ is a link connecting port 1 of node v with port 2 of node u.

The network is defined as a pair (V, E), where V is a set of nodes and E is a set of links $E \in 2^{C_V \times C_V}$ (2^X is a powerset of a set X, and $\times$ is the Cartesian product).

Network path is a sequence of links $[(s_i, d_i)]_{i=1,\ldots,k}$ such that $d_i = s_{i+1}$ for all $i = 1, \ldots, k-1$. If for each i, s_i is a pair of connectors (v_i, p_i) and $d_i = (u_i, q_i)$, then we call v_1 the *source node* of the path and u_k the *destination node* of it. The current implementation non-restrictively assumes that the path from A to B is the same as the path from B to A.

What happens when agents running on some hosts $A \subset V$ report their measurements? First of all, using our path detection module, we extract a path (as defined above) from each agent until the target node $t \in V$. This means that for each agent $a \in A$ we obtain path p_a with source node a and destination node t. Also, given the measurement $m(a, t)$ for the host t reported by this agent a and a threshold θ, we obtain a simple classification function $f : \{p_a\}_{a \in A} \mapsto \{0, 1\}$ such that $f(p_a) = 0$ if $m(a, t) \leq \theta$, and $f(p_a) = 1$ otherwise. Paths p having $f(p) = 0$ are considered trouble-free and all links along those paths are whitelisted.

We base our troubleshooting application on two assumptions. First, every underperforming path contains at least one link causing the problem. Second, we assume that in the case of a problem, it is most probable that exactly one link is causing it, but we shouldn't exclude the case that multiple links are underperforming at the same time. Having this intuition, we assign the weights to the links pointing to the most probable points of failure: the bigger the weight, the more critical the probable link issue.

Let R ("Red") be the set of underperforming paths, i.e., $R = \{p : f(p) = 1\}$ and G ("Green") be the set of problem-free paths, i.e., $G = \{p : f(p) = 0\}$. Now, we can simply count, per each link in every path in R, the number of underperforming paths that include it and then normalize w.r.t. the total number of the underperforming paths (in this way the maximal possible weight will be 1). Also we should "whitelist" the links (by giving them weight 0) from the paths of agents that returned acceptable measurements. Then the weight is defined as ($|R|$ is the cardinal of set R):

$$w(l) = \begin{cases} \frac{|\{p \in R : l \in p\}|}{|R|}, & \text{if } \exists p \in R : l \in p \text{ and } \nexists q \in G : l \in q. \\ 0, & \text{if } \exists q \in G : l \in q. \end{cases}$$

As the output of our trouble localization procedure, we obtain weights (via function w) for links suspected of network issues. In short, the worse the link underperforms, the higher its weight; if the link is on a problem-free path, the path is safely ignored; for other links we have no infromation.

2) Extracting network topology: Topology discovery is a base network service that is typical for control platforms. The **Topology** module of ODL (in the Hydrogen release) provides the list of all links between the switches together with their properties. The **Switchmanager** module can provide the list of all switches and their attributes, while the **Hosttracker** module has an up-to-date database of the active hosts in the network. From the latter module we request the such information like a given host's MAC and IP addresses, and the switch and its port to which the host is connected. By merging the information from these modules, we were able to obtain a complete picture about the topology and display it to the SDN-RADAR's user.

3) Path extraction: In order to locate a network problem for a given type of traffic, it is crucial to know what exactly happens to the packets. More precisely, we are interested in the *path* taken by each packet: which switches it traverses and which links it goes through. Unfortunately, there is no direct and easy way to trace a packet in OpenFlow-based networks. The problem of tracing packets in SDN is a hot research topic. For example, in [17] the authors proposed to use test packets with a special marker which is written in a header field and trace such packets using additional high-priority rules installed to the switches. The method requires a packet header field is not used by the current network setup and also requires to tag switches in a special way that can impose constraints on a minimal size of such field. A related idea is used in [18], where an unused field is employed to carry a precalculated (based on the topology) code which can be decoded to show the full path that the packet took. Still, this approach may be problematic for large networks.

In the proof of concept we assumed that the routing rules are installed only by the default sample routing application of the OpenDaylight controller (Hydrogen release), called SimpleForwarding. Once a new host H appears in the network, the SimpleForwarding application add in each switch a rule that matches the destination IP of a packet with the IP address of H, and forwards the packet to the next switch in the shortest path to H. The last switch in the shortest path also sets the packet destination MAC to the target H's MAC before forwarding the packet to host H.

The knowledge of the rule templates used by the SimpleForwarding application allowed us to easily detect the next hop. That is, if we want to know which path a packet will traverse from host A with destination IP of host B, we look first at all rules of the switch connected to A and find one which would match such packet. This rule will tell us to which port the packet is outputed and we can see (using the topology module) to which next switch that port is linked. By continuing this procedure until reaching host B, we end up with a sequence of connectors (pairs switch-port) representing the path we were looking for.

The advantage of the SDN-RADAR approach is that it does not produce any modifications on the switches in contrast to the methods mentioned above, and only requires a read permission. This is an important benefit especially if the tool is used in an already set up and running network.

4) Functional tests: In our network with running agents, target host and OpenFlow switches connected to the ODL controller, Mininet allows to customize a given interface with the *tc* tool. In particular, we can inject an artificial delay into network links[2]. We used this functionality to artificially delay traffic on one link and run our troubleshooting tool. Indeed, the result took the delay into consideration. The prototype reported two links with weight 1. One of these two links was actually slowed down by us, and thus was underperforming, so the algorithm correctly localized the possibly underperforming link. To pinpoint the exact underperforming link we just needed to deploy an additional agent. This means that the more agents are used, the more precise the result of our algorithm.

VI. Discussion and future work

This work presents SDN-RADAR, a novel approach and tool for network troubleshooting where user-side performance measurements are first-class citizens. The aim is to assist network administrators in finding out the cause of network issues based on (i) service degradation metrics as a starting point of the troubleshooting process, and (ii) SDN management features such as runtime network topology and forwarding device state. This approach has several advantages: it does not incur big instrumentation costs, and it works across different locations, business domains and service delivery protocols. Moreover, as we have shown in our proof of concept, the approach allows for an implementation that does not require to introduce any changes in the running network configuration. As a major contribution, we developed the first proof of concept that shows that with our approach it is possible to monitor, at runtime, the paths packets take into the network and which are the underperforming network links (i.e., paths experiencing congestion or packet drops).

Our algorithm considers network paths as the domain of search for network issues, and these issues relate to user experience. Bad user experience (e.g., slow or non-updating frontend, errors when loading a web page) can be an indicator of network faults such as broken links, network misconfigurations and misbihaving switches. SDN-RADAR does not perform root-cause analysis; it is intended, rather, to be a warning system from the perspective of the user about a network fault that affects service delivery. SDN-RADAR can be complemented by solutions that address the correlation between service degradation and affected component (DNS, transport, etc). Apart from its reduced cost as opposed to existing vendors', SDN-RADAR has the advantage that it directly ties into business metrics. Enterprise management can easily understand and reason in terms of user experience, and that is the starting point of the measurements that are part of the troubleshooting process. Of course, for accurately understanding user experience, one should build a model for different types of user interactions with the service that would better represent user population from various locations. We plan to generalise our path detection technique, and test SDN-RADAR in a large-scale production-ready environment and with different topologies as our next steps.

[2]See the Network Emulation Utility http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

References

[1] "Emerson Network Power, Ponemon Institute Study Quantifies Cost of Data Center Downtime," hhttp://emersonnetworkpower.com/en-US/Brands/Liebert/Documents/White%20Papers/data-center-costs_24659-R02-11.pdf, 2014.

[2] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic Test Packet Generation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. ACM, 2012, pp. 241–252.

[3] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarifis, and P. Kazemian, "Leveraging sdn layering to systematically troubleshoot networks," in *HotSDN '13*. ACM, 2013, pp. 37–42.

[4] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *CoRR*, vol. abs/1406.0440, 2014. [Online]. Available: http://arxiv.org/abs/1406.0440

[5] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007.

[6] A. Anand and A. Akella, "Netreplay: A new network primitive," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 14–19, Jan. 2010.

[7] Y. Zhuang, E. Gessiou, S. Portzer, F. Fund, M. Muhammad, I. Beschastnikh, and J. Cappos, "NetCheck: Network Diagnoses from Blackbox Traces," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 115–128.

[8] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 55–60.

[9] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 71–85.

[10] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling Record and Replay Troubleshooting for Networks," in *Proc. 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. USENIX Association, 2011, pp. 29–29.

[11] G. Gheorghe, "Secure Election Infrastructures Based on IPv6 Clouds," http://www.gen6-project.eu/fileadmin/user_upload/5_Booklet_Secure_Election_print_plain.pdf, 2014.

[12] Nephos6 Inc., "Sonar Solution Overview," http://www.nephos6.com/pdf/Sonar-Brochure.pdf, 2014.

[13] F. Jacob, B. Joris, S. Lepage, J. Dusart, and J.-M. Frère, "Role of the conserved amino acids of the SDN loop (Ser130, Asp131 and Asn132) in a class A beta-lactamase studied by site-directed mutagenesis," *Biochem. J*, vol. 271, pp. 399–406, 1990.

[14] T. Avanesov, G. Gheorghe, M. Palattella, M. Kantor, C. Popoviciu, and T. Engel, "Network troubleshooting with sdn-radar," in *accepted as a demo to Integrated Network Management (IM 2015), 2015 IFIP/IEEE International Symposium on*, 2015.

[15] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. ACM, 2010, pp. 19:1–19:6.

[16] "OpenDaylight project," http://www.opendaylight.org.

[17] K. Agarwal, E. Rozner, C. Dixon, and J. Carter, "Sdn traceroute: Tracing sdn forwarding without changing network behavior," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 145–150.

[18] H. Zhang, C. Lumezanu, J. Rhee, N. Arora, Q. Xu, and G. Jiang, "Enabling layer 2 pathlet tracing through context encoding in software-defined networking," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 169–174.